An analysis on Version Control Systems

Conference Paper · February 2020

DOI: 10.1109/ic-ETITE47903.2020.39

4 authors, including:

N. Deepa
Pondicherry University
55 PUBLICATIONS **3,444** CITATIONS
SEE PROFILE

Prabadevi. B
M S Ramaiah University of Applied Sciences
82 PUBLICATIONS **3,678** CITATIONS
SEE PROFILE

Krithika Lb
Vellore Institute of Technology University
25 PUBLICATIONS **377** CITATIONS
SEE PROFILE

An analysis on Version Control Systems

N.Deepa
School of Information Technology and Engineering
Vellore Institute of Technology
Vellore, India
deepa.rajesh@vit.ac.in

B.Prabadevi
School of Information Technology ad Engineering
Vellore Institute of Technology
Vellore, India
prabadevi.b@vit.ac.in

Krithika L.B
School of Information Technology and Engineering
Vellore Institute of Technology
Vellore, India
krithika.lb@vit.ac.in

B.Deepa
Department of Information Technology
Sairam Institute of Technology
Chennai, India
deepa.it@sairamit.edu.in

2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE) 978-1-7281-4142-8/20/$31.00 ©2020 IEEE 10.1109/ic-ETITE47903.2020.39

***Abstract*—Managing the source code of the project and other related documents in an organization is a mandatory need, which may ensure clarity in the delivery of the product enhancing the focus of the organization towards its intended product's quality. In this digital era of computing, we have many software configuration management tools to handle various documents, its revisions, versions and so. In this paper, we analyze the importance of various Version Control Systems (VCS) evolved to assist the software development lifecycle of the project, and compare favourite VCS tools in the market based on their features, measure their performance across chosen attribute. Also, we propose a new tool having some of the best features found in our comparison study as well as a few extra attributes that we believe will raise the quality of this new tool. This tools can combat the issues we face with existing tools in the market.**

Keywords— version control systems, software configuration management, checkpoint, GIT, cryptography

I. INTRODUCTION

Version control, which is considered to be a very important component of software development life cycle, which can also be called as revision control or source code control, is the management of changes to documents, computer programs, large websites, and other collections of information. Changes can be identified in some ways, such as "revision number", "revision level" and "revision" etc.

People prefer using such VCS, which can manage teamwork with no difficulties, as that is mandatory when coming to big projects. They can either run as standalone applications or can also run by embedding them in various types of software such as word processors and spreadsheets(like google docs) and in various types and variants of CMS(like Wikipedia's Page history). Using VCS, it is also possible to perform various operations like edits tracking, mistake correction, and vandalism defending and spamming protection.

In software engineering, revision control's primary aim is to provide control over changes made to source code. Its functionalities can also be enhanced, and it can be made use of to make electronic documentation too.

As complex teams are involved in the modern software development, it is very frequent for many versions of the same software to be deployed in different locations or systems and for them to be working synchronously on its development. Most of the times only a specific part of the program does contain bugs or features (by the fixing of some problems and the introduction of others as the program develops). Therefore, to locate and to fix bugs, it is of absolute important to be able to retrieve and run various versions of the software to determine in which version(s) the problem occurs. It may also be necessary to develop different parts of the software in parallel (for instance, where one version which has all its bugs fixed, but no new features (branch), while the other version is where new features, which is still under testing are worked on (trunk).

At the simplest level, going by the traditional way, we can retain multiple copies of the different variants of the program, and label them appropriately. It is manual version control. Formerly it was used on large software projects. While this method can work, it needs much work, and it is incredibly inefficient. It needs the humongous amount of discipline and can also create new mistakes on the part of the developer. As during all these works codebase tends to remain the same, it also involves creating an admin user who is responsible for granting privileges to who can configure which variant etc. So that the code base is not compromised no matter what, and in this way, the immense amount of complexity gets added. To overcome all these, we have got many VCS through which the version control management steps are secluded from the vision of users of the software.

Moreover, not only in software development but other environments too whichever stream we go, it has got common nowadays for a single document to be controlled by a team, the members of which may not be present physically together at a place and may have different tasks on the whole. VCS that has the capability to imprint and to record for ownership of rework to documents may be extremely of great help in such situations.

The widespread use of computers to automate tasks in diverse fields pushed the growth and development of version control systems and hence became a vital part of the software

configuration management practice. This made it possible to achieve real-time collaboration regardless of time and place and became a safety net for the entire project.

Many version control systems exist today and while the older ones had a steeper learning curve due to their command-line nature and added weight on the brain to understand the commands as well as the architecture of the VCS, many modern VCS tools lessen the burden by providing an intuitive CLI with extensive documentation and help material built-in (e.g. Git) and even comes with easy-to-use GUI tools or integrates with the system's default file browser (e.g. Subversion and TortoiseSVN).

These tools also come in two architectural types—centralized or distributed. Centralized relies on a single repository to store the bulk of project data as well as versioning data and client interfaces only fetch the latest version to work with. Distributed systems—which are the current trend—put a copy of the entire repository on each user's system which makes it easy to work independently and offline.

II. Study of Existing Version Control Systems

We studied several version control systems in use currently which cover different approaches that went in building them such as centralized vs distributed architectural types. We have also included some older tools in the study to identify how far their modern counterparts have come regarding performance, usability and interoperability. Specifically, we studied the following eight tools:

- Revision Control System (RCS)
- Concurrent Version System (CVS)
- Perforce Helix
- BitKeeper
- Subversion
- GNU Bazaar
- Git
- Mercurial

A. Revision Control System (RCS)

RCS, one of the very first VCS, which came into existence in 1982 by Walter F. Tichy who was from Purdue University [1]. RCS is currently kept up by the GNU Project. RCS was initially developed for software development, but it is also useful for text documents which need frequent revision. RCS can only do single file operation[2]. As of now, it does not have a provision to support atomic commit. It supports branching though for individual files, but the syntax of its operation is hard to handle. Besides using branches, teams prefer to use the built-in locking mechanism and work on a single head branch. It has got straightforward to understand the structure. However, the main con being its operational limitation is restricted to only one user at a time. No concurrency. It is only capable of working locally. This tool can only support the waterfall model of the software development workflow[3-5]. No GUI alternative is available.

B. Concurrent Version System (CVS)

Dick Grune developed CVS as a series of shell scripts. CVS uses a client-server architecture model[6]. Released under the GNU GPL. It was very popular with open-source projects in its early days. No new releases since 2008 or put in other words, is dead. CVS uses a client-server architecture. Client and server may be run on the same machine for local development. Supports concurrent development where each developer edits his/her working copy and checks-in the changes to the server. All files are versioned with a number which is incremented with further check-ins. We have got a feature which allows the users to different versions, a complete history of changes can be viewed and analyzed, besides being able to check out an old snapshot of the project sorted by a given date or as of a revision number. CVS servers can allow "anonymous read access". Can maintain different branches for a project [7]. CVS can store various versions of the same file using a unique feature known as delta compression. CVS does not consider the move and renaming as separate versions. GUI clients such as TortoiseCVS and SmartCVS available.

C. Perforce Helix

Perforce Helix is one of the leading proprietary revision control systems which was developed by Perforce Software. It typically uses a client-server model for managing projects[8-12]. [9] provides asset management and protection for corporates. Can make content available as Git repos or for other clients. Salesforce, VMWare, SAP, Samsung etc. use Helix for SCM. Complete history and metadata of file maintenance including history for branching, renaming, moving, copying, and deleting files. Extensive support for merge tracking and re-merging prevention. The advanced administration is made possible using graphical tools. Mostly its mode of operation is centrally, but it also supports distributed mode of operation. Logical changes tracking is possible for changed files. Support for ASCII, Unicode, binary, symlinks. Internationalization, localisation, RCS-style keyword expansion, File compression, Network transfer, Server-side event triggers, command-line client and extensive API's. Backup is also supported extensively. A sort of local administrator who is also called a Broker is responsible for various administrative tasks like local policies implementation, available commands restrictions, load sharing administration tasks. Backing files to different storage to get back main server disk space. Encrypted SSL connections from clients to the server. Efficient team management and protection. Perforce Helix comes with online hosting with upon purchase where one can host repos and collaborate with a team.

D. Subversion

Subversion was developed by CollabNet. It is a centralized revision control system[13]. It is open-source under Apache license. Second-most favourite VCS in use after GIT. Currently an Apache-funded top-level project. A variety of top organizations in the world like Source Forge, Apache Software Foundation, GCC, Mono, FreeBSD etc. uses SVN for source control. Subversion supports branching and tagging to support non-linear workflows. Branching is

not a very costly operation like other VCS. No matter what if any simple change also made, the files get to retain full revision history. Commits as true atomic operations. Natively client-server, layered library design. Language bindings and support are available for a wide variety of languages. Single source of access control. Merge Subversion also supports tracking feature. Change lists to organise commits into commit groups. Mature user-interfaces available (TortoiseSVN). Deveo, Assembla, RiouxSVN offers reliable online hosting of subversion repositories [14-16].

E. GNU Bazaar

GNU Bazaar was made by Canonical, the same team behind Ubuntu. It is both distributed and client-server. It can be used by a single developer working on multiple branches of local content [17-19]. Written in Python and cross-platform. Free and open-source. Used by Linux Foundation, Ubuntu, MySQL, Debian, MariaDB and many more. Has support for true branching which is also cheap. Supports code management in both the central and distributed modes with a great amount of ease. Very easy to switch from Subversion because of CLI similarity. Intuitive CLI with extensive documentation as well as inbuilt GUI tools. Is flexible and can adapt to a lot of different workflows. It has got a very effective performance on large trees, despite being on slow networks even if the project has got an intense revision history. Provides rename tracking for both files and directories. Does not need a dedicated server. 100+ plugins available and you can make your own with the Python API. Launchpad integration for online hosting of Bazaar projects [20].

F. Git

Git is an advanced version control system (VCS). Its functionalities include working on files and to coordinate teamwork on those files among multiple people and to trace changes. Git was initially created by Linus Torvalds in 2005 for, with a goal to support the Linux kernel's development. It has an open-source license of GNU General Public License version 2.0 [21-23]. In the current day, it is the most widely used VCS these days in companies, besides it being extensively popular in the standalone developer community. As of 2016, it enjoys hefty 70% of all the search interest among VCSs and has most questions per day in Stack Overflow. It has got wide extensive support of the open-source community, consisting of quality developers behind. It supports MS Windows and all the Unix-like systems. It supports both linear and nonlinear development. It can be used with both traditional software models(waterfall) and modern software models(agile). Branching, committing, undo, checkpoint, merging all these complex operations are supported with great ease in GIT, which gives the user a bigger reason of why to use Git because all these features are very much needed in complex software projects. Git has a distributed repository type. The cryptographic integrity of every bit of software project is very well taken care of by Git. Every file and commit is checksummed. For people who are not familiar with git style of working or git's typical workflow, it supports various kinds of workflow like, Dictator and Lieutenants, Integration Manager, Subversion-style workflows etc. It is suitable for almost all kinds of projects varying from the size in KB's to EB's or any IDE. Easy to use CLI. GitHub, Bit Bucket, GitLab are few of the popular online hosting providers for Git repos.

G. Mercurial

Matt Mackall developed mercurial and implemented in Python [24-28]. One of the most popular VCS tools available. Decentralized, fully-distributed architecture. It's widely used by various companies like Facebook, Mozilla, Nginx, and NetBeans. Also used in Mozilla and Octave. Cross-platform support. It has a license of GNU GPL v2. Supports decentralized, fully distributed collaborative development. It has got a very higher and progressive merging and branching capabilities compared to other VCS. No concept of permissions needed. For this basically, we create a repository that is accessible by members of a certain user group. Each commit is identified by a hex string, which is nothing but cryptographic hash of data. Has a very intuitive CLI. Quite a few GUI clients are available including TortoiseHg which is well-built and user-friendly. Many online hosting options—CodePlex, Assembla, BitBucket, RhodeCode etc.

H. BitKeeper

BitKeeper was developed by BitMover Inc. One of the first distributed versioning tools. Originally a proprietary software. Made open-source on 9th May 2016 [29-34]. Was used for managing the Linux project. Is currently used by only a tiny number of projects and companies. Support for huge projects. Has minimal footprint and is very fast. True file/directory renaming support. Inbuilt cross-platform GUI tools for committing, merging and for diffs. All changes in BitKeeper are reproducible snapshots. True offline support. Developers can work peer-peer without having to involve a "master" copy. Completely hackable and can be configured to run tasks before or after an event occurs. Automatically checks the integrity of the system to alert the user. BitMover provides an online hosting platform for BitKeeper projects called BkBits.

III. Proposal of a Postmodern Version Control System

We propose a wholly new Version Control System which not only incorporates the significant strengths of all the version control systems we surveyed and analysed in our comparative study but also a few features that lessens the need for third-party tools and suits modern workflows better. We strictly studied and analyzed all the tools based on the following key parameters:

1. Repository Type
2. Popularity
3. Access Control
4. Security
5. Storage Model
6. Ease of Use

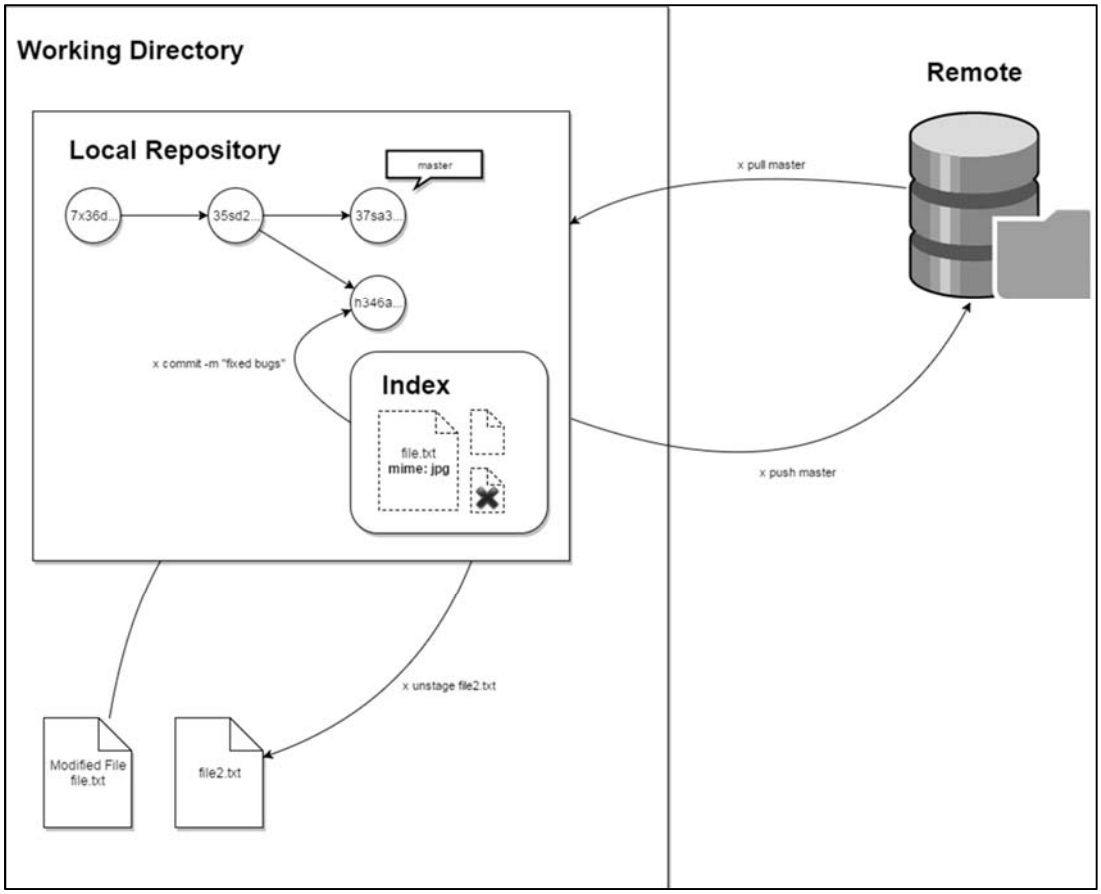

Fig. 1. Architecture diagram for the proposed tool.

In the following sections, we will break down all the new features of the proposed tool based on the above parameters. Note that we are going to refer to the proposed tool as 'X' in some places.

A. Repository Type

This essentially refers to the repository type we are going to use for our new tool. Whether it is centralized or distributed, its support for different workflows and flexibility of repository.

1) Distributed architecture with an option for a centralized approach—The repository architecture type will be distributed which will be similar to that of Git to allow parallel workflows. Hence, there will be three main components that will resemble an X repository—a working directory, a staging area and the repository itself. A 'bare' option will be present to create a repository without any working directory instead.

2) Source code written in portable Rust—This will simplify development and will enable us to target multiple platforms at once. Rust is a very modern low-level language which is aimed as a replacement to C. The Rust community is very active, and the language is powerful and efficient. Note that this is just the core part of the system we are talking about. The GUI components will be separately maintained as true cross-platform GUI is very hard to build and maintain. However, we will utilize cross-platform GUI tools to keep the code consistent as much as possible.

3) Remote operations through HTTPS—HTTPS is increasing in popularity day-by-day and is very secure and widely used in the web. Henceforth, instead of trying to reinvent the wheel and come up with a protocol for our own, all remote operations will be done via HTTPS only.

B. Popularity

Organizations using this tool in addition to online hosting support for the tool. Moreover, support for other tools.

1) Import/Export support for popular repository formats—In order to make people switch expeditiously to the proposed system, we need to make it compatible working with popular repository types. Options for importing and exporting to repositories of popular formats will be available to assist in switching. Based on the nature of the repository—centralized or distributed—a bare or a normal repository will be generated when attempting to import commits or changesets from another repository format. While this will add some overhead to the overall functionality, this is

mandatory so that enterprises and professionals won't be burdened with the task of converting.

C. Security

Security parameter relates to the integrity of the revision history, encryption of the revisions and major flaws discovered recently.

1) HTTPS for remote communications and OAuth for authentications—We discussed the purpose for making HTTPS the primary protocol for remote operations in regards to the architecture. However, HTTPS is also normally very secure in nature and is the current recommendation in the web community. In light of this, we decided to adopt HTTPS for incoming/outgoing traffic between two repositories as it is safe, efficient and compatible with many other applications. There's no need to write programs and APIs for working with a new protocol. It is also known that many online repository hosting sites provide access through SSH and OAuth which are two secure ways of accessing remote content across the internet. While SSH is ubiquitous and secure, we believe it as a power that carries way too many responsibilities and is not easily understood by many. However, HTTPS and OAuth are simpler schemes and also share similar properties. With the advent of HTTPS/2, the traffic is getting faster and safer. This approach is thus reliable and provides safety to the user and reduces the complexity of understanding the underlying mechanism.

2) SHA-2 encrypted commits—Commits will be stored in the repository by calculating the SHA-2 checksums for the files to be committed. SHA-2 is currently not broken and is a very fast and strong encryption algorithm. This will ensure the safety of the commit messages when transferred.

SHA-1 was theoretically broken as long back as 2005 and was officially deprecated by NIST in 2011 [35]. Many organizations have since switched to the safer SHA-2 method, but a lot still employ the older, unsafe predecessor. In February 2017, a team of Google researchers published a paper describing a faster technique of cracking the algorithm and achieving the same SHA-1 checksum on two colliding files.[The researches also stated that Git is vulnerable to this and Subversion is already facing issues when colliding files are present in the repository.

D. Access Control

It included all the mode of permission control, types of permission control and user management.

1) Optional lock/unlock support for the repository—Locking is not a common feature in the distributed version control system. The merged model is preferred overlocking. However, locking can sometimes make the process simple and can help new developers get acquainted and started working with the rest of the team. They will not need to worry about making conflicting changes or changes they were not supposed to make. The proposed tool will allow the user to turn on locking optionally to permit the key developer to lock access to certain areas of the repository or certain files to certain users. Again, the management part can be easily achieved either through the CLI interface or through the GUI tool.

E. Storage Model

Storage model takes into account change set/snapshot, disk space consumed and handling of binary files.

1) Improved binary file support—A common problem we observed with the internals of the popular VCS tools are how they deal with binary files. Usually, their designs treat the files equally and have no idea about exactly what type of file they are. While this doesn't add constraints as to what operations can be performed since every operation is supported on theoretically all files, these are usually slow because operations like diffs and calculating deltas while are faster when dealing with text files are usually slow when done in the same way in binary files. There exist several different algorithms to make this more efficient but since a VCS tool has no idea about the exact binary format, these operations tend to take time. What we propose is to add attribute data to these files to store filetype information when committing. This way, we can implement faster algorithms for the common operations which are faster when dealing with particular types. This of course has a lot of implications but is indeed possible at the cost of a larger size for the VCS itself. While there are so many binary formats present, we aim to implement specific logic for common formats like image data or audio data. We also propose a way to extend this mechanism so that developers can add custom logic for other types of binary files to suit their specific needs.

Another common behaviour is that these tools load the binary file entirely into the memory and perform a diff. While this is fine for smaller files, the operation could block for larger files. This can be sped up by splitting the large files or larger deltas to smaller chunks and diffing these smaller pieces and combining the result. This is another strategy that we propose to be included in our tool.

Together, these changes can fasten up VCS operations on binary files and can efficiently handle larger files and larger repositories.

2) SHA-2 encrypted commits—We covered this previously but to reinforce the idea behind the choice based on the current context is important. While encrypting commits in SHA-2 has its upsides when it comes to security, it is also important when it comes to preserving the integrity of the repository. This means that our system can easily identify the bits as they are in transit and make sure that the exact some bits are received at the other end and are all in one sequence thus eliminating data corruption possibilities. As long as one doesn't mess with the internal files, the system is incomplete awareness of what files are present and their history.

F. Ease of Use

This parameter takes interest in UI/UX of the programs. The availability of GUI clients. Moreover, the interactiveness of the interfaces.

1) Clean and consistent Command Line Interface (CLI)—While Git is currently the most popular VCS tool in use, one thing we found odd was that it had a messy CLI. Across many discussions, it was found that one reason why people particulate dislike Git was because its CLI was hard to understand. This is true particularly for people coming from another tool like SVN due to the difference in the meanings of commands. However, there were some inconsistencies in the name and purpose of some commands and what they actually did. For example, resetting some files or all files in the working directory is a common operation in Git and `git reset` is the command you use to achieve this. To revert every file in the working directory to the last committed state, this is what is done:

```
$ git reset --hard
```

However, if you wish to reset a single file to its last committed state, you have to use a different command:

```
$ git checkout file.txt
```

This is very confusing for first-timers as well as seasoned coders. Similarly, the git reset command is also used to unstage files from the working directory. The reset command so far is ambiguous in terms of purpose.

The goal of this tool is to resolve all those inconsistencies and lack of clear usage we found in popular tools and provide a clean, modern and welcoming interface that is easy to remember and fun to use. Even if it means adding a few extra commands, we believe explicit is better than implcit. Instead of having one single command which is ambigious, our tool will provide different commands which are named closely related to their purpose:

```
$ x unstage file.txt
$ x reset file.txt
$ x reset --all
```

2) Inbuilt cross-platform GUI tool

Subversion scores highest in our *Ease of Use* analysis because it provides a lot of mature user interface options and comes with a neat GUI tool which is easy to navigate and use. While on the long term, learning to use and muster CLI could prove productive, inclusion of GUI tool is necessary for novice users to work with the system. This is why the new tool will have a built-in cross-platform GUI client. Needless to say, the client will incorporate full functionality provided by the CLI as well as allow other developers to write plugins to extend the functionality of the application. One neat thing we could do is to provide an interface to an online repository of plugins to the GUI client as well as extensions to the core tool (like how we discussed about extending binary file support) so that users can quickly find and incorporate new community-driven features.

3) Filesystem integration for popular platforms

A neat feature that TortoiseSVN, a subversion client provides is how it integrates with the GUI filesystem explorer of Windows. This makes it even easier to perform VCS operations just by right-clicking a file or by accessing the top ribbon menu to find a layout of supported operations. It also displays state of the repository with colors and icons. This is very good UX.

Hence, in addition to the GUI client that's built-in, users can also install opt-in filesystem integrations for their platforms to provide a similar experience. There will be integration projects on popular platforms such as Mac, Windows and Linux. These integrations will also be developed with extensibility in mind so that developers can add more functionality and tweak the experience to suit their styles and workflows.

IV. Parametric Comparison of Existing Tools and the Proposed Tools

To better understand how the tools perform, we tabulated some important characteristics of each tool along with the proposed tool [35-42] in *Table 1*.

We present a graphical comparison of the tools we studied and the new tool we proposed. This comparison is performed by analyzing the 6 key parameters we considered for comparison. Since there aren't any quantitative parameters, the tools are scored on a scale of 0-10 for all parameters. While the proposed tool is still theoretical, we also included it in the comparison and conducted another unbiased comparison to assign scores to it. The results were graphed and are included in the figures Fig. 2 to Fig. 6.

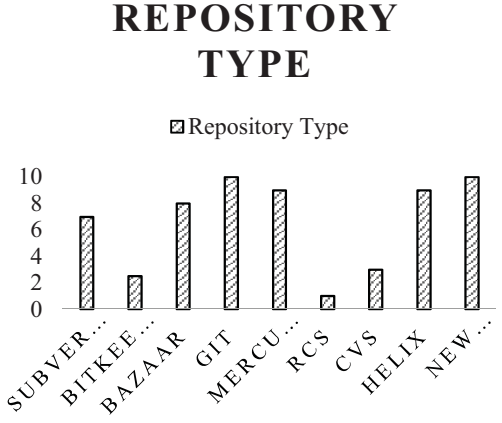

Fig. 2. Repository Type

The ranking criteria for the repository type are :

- Centralized repository, distributed repository or can support both
- Programming language used to create the tool
- Remote control of repository available through HTTPS

POPULARITY

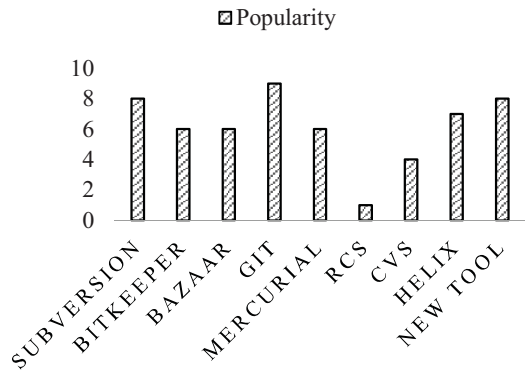

Fig. 3. Popularity

The ranking criteria for popularity is:

- The user base of the tool
- Rating by experts
- Ability to support other popular tool repositories

ACCESS CONTROL & SECURITY

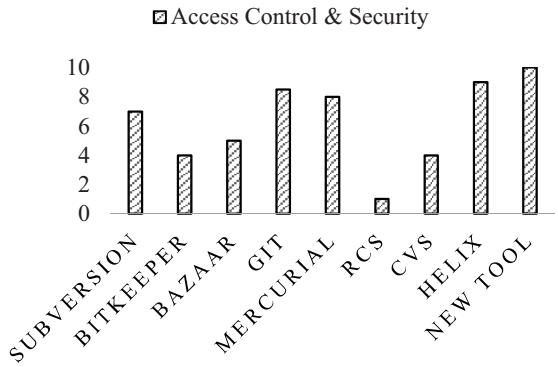

Fig. 4. Access control and security

The ranking criteria for access control is:

- Type of connection used for remote communication like HTTPS or HTTP
- Authentications implemented in the tool like OAuth
- Commit ID creation

EASE OF USE

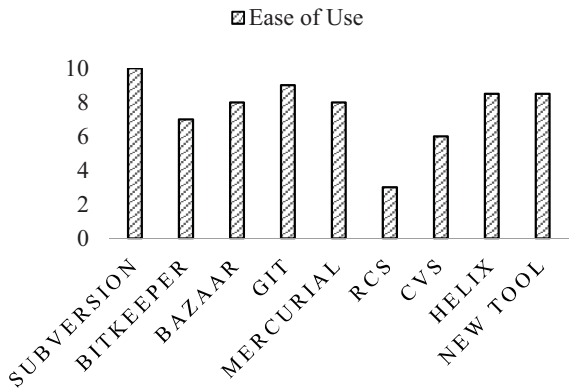

Fig. 5. Ease of use

The ranking criteria for ease of use is:

- Inbuilt cross-platform GUI tool
- Understandability and accessibility of GUI elements
- Clean and consistent Command Line Interface
- File system integration for popular platforms

STORAGE MODEL

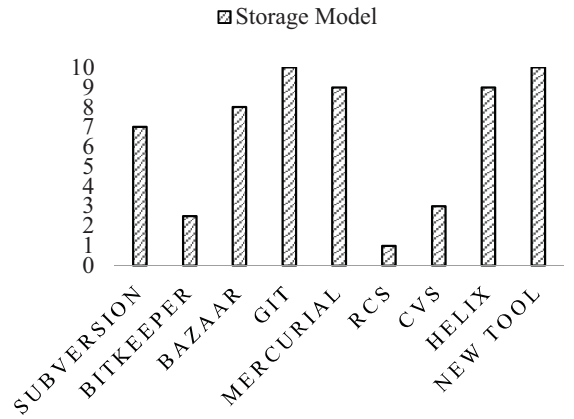

Fig. 6. Storage Model

The ranking criteria for storage model is:

- Binary file support
- Time is taken for committing a file
- Time is taken for retrieving a file
- Storage security measure

TABLE I COMPARISON OF VARIOUS TOOLS WITH PROPOSED TOOL

	SUBVERSION	BITKEEPER	GNU BAZAAR	GIT	MERCURIAL	RCS	CVS	HELIX	PROPOSED TOOL
Repository type	Client server	Distributed	Distributed and client server	Distributed /Centralized	Distributed	Local	Client-server	Client server	Distributed/Centralized
License	APACHE	APACHE	GNU GPL	GNU GPL	GNU GPL	GNU GPL	Client server	Proprietary	MIT
Platforms supported	Unix-like, Windows, OS X	Unix-like, Windows, OS X	Unix-like, Windows, OS X	POSIX, Windows, OS X	Unix-like, Windows, OS X	Unix-like	Unix-like, Windows, OS X	Unix-like, Windows, OS X	Unix-like, Windows, macos, Linux
Cost	Free	FREE	FREE	FREE	FREE	Free	FREE	PAID	FREE
Storage method	CHANGESET AND SNAPSHOT	CHANGESET	SNAPSHOT	SNAPSHOT	CHANGESET	CHANGESET	CHANGESET	CHANGESET	SNAPSHOT
Scope of change	TREE	TREE	TREE	TREE	TREE	FILE	FILE	TREE	TREE
Revision ID's	Numbers	Changeset keys, numbers	Pseudorandom	SHA-1 hashes	Numbers, SHA-1 hashes	Numbers	Numbers	Numbers	SHA-2 hashes
Network protocols	Custom (svn), custom over ssh, HTTP and SSL	Custom, HTTP, rsh, ssh, email	Custom over ssh, custom over HTTP, HTTP, SFTP, FTP	Custom (git), custom over ssh,[17] HTTP/HTTPS, rsync, email, bundles	Custom over ssh, HTTP, email bundles	File system	Pserver, ssh	Custom	HTTPS
Souce code size	41 MB	99 MB	4.1 MB	23 MB	20 MB	5.3 MB	10.3 MB	Unknown	Unknown
Atomic commits	Yes	YES	YES	YES	YES	NO	NO	YES	Yes
Signed revisions	NO	UNKNOWN	YES	YES	YES	NO	NO	YES	Yes
Merge tracking	YES	YES	YES	YES	YES	YES	NO	YES	Yes
International file support	YES	UNKNOWN	YES	YES	YES	NO	UNKNOWN	YES	Yes
Support large repos	YES	YES	NO	PARTIAL	PARTIAL	NO	NO	YES	Yes
Interactive commits	YES	YES	YES	YES	YES	NO	NO	NO	Yes

V. Conclusion

Hence after going through best version control tools available in market, though we can say that GIT is currently the best tool available in market followed by apache subversion and mercurial and for enterprise usage perforce helix being the best all tools have got its set of flaws including GIT which cannot conclude to be as the best tool. Hence, we have proposed a new tool which contains the best feature from each of the tools in addition to a few extra features too to support modern workflow. In future, this tool can be implemented and brought into use.

References

[1] RCS GNU Project - https://www.gnu.org/software/rcs/

[2] Tichy, W. F. (1985). RCS—a system for version control. Software: Practice and Experience, 15(7), 637-654.

[3] RCS tutorial - http://archive.oreilly.com/pub/a/perl/excerpts/system-admin-with-perl/five-minute-rcs-tutorial.html

[4] RCS a system for version control - http://dl.acm.org/citation.cfm?id=4202

[5] RCS in brief - http://jodypaul.com/SWE/RCSTutorial/RCSTutorial.html

[6] CVS wikipedia - https://en.wikipedia.org/wiki/Concurrent_Versions_System

[7] Using CVS - https://www.ibm.com/support/knowledgecenter/SSSHYH_7.1.0.6/com.ibm.netcoolimpact.doc/admin/imag_selecting_CVS_version_manager.html

[8] What is perforce helix - https://www.perforce.com/versioning-engine

[9] Perforce manual - - https://www.perforce.com/perforce/r15.2/manuals/dvcs/

[10] Introducing helix - https://www.perforce.com/blog/150303/introducing-helix

[11] Perforce helix - https://en.wikipedia.org/wiki/Perforce_Helix

[12] Enterprise perforce - https://softwareengineering.stackexchange.com/questions/85845/why-big-companies-use-perforce

[13] Apache subversion wikipedia - - https://en.wikipedia.org/wiki/Apache_Subversion

[14] Apache subversion - https://subversion.apache.org/

[15] Subversion wikipedia - https://en.wikipedia.org/wiki/Subversion

[16] Version control with subversion - http://svnbook.red-bean.com/

[17] Bazaar - http://bazaar.canonical.com/en/

[18] Bazaar vs git - http://wiki.bazaar.canonical.com/BzrVsGit

[19] Bazaar stackoverflow - http://stackoverflow.com/questions/14926774/what-is-the-state-of-bazaar-version-control

[20] GNU bazaar - https://en.wikipedia.org/wiki/GNU_Bazaar

[21] Git documentation - https://git-scm.com/doc

[22] Github git - https://github.com/git/git-scm.com

[23] Git wikipedia - https://en.wikipedia.org/wiki/Git

[24] Mercurial - https://www.mercurial-scm.org/

[25] Mercurial wikipedia - https://en.wikipedia.org/wiki/Mercurial

[26] Mercurial vs git stackoverflow - http://stackoverflow.com/questions/35837/what-is-the-difference-between-mercurial-and-git

[27] Managing with mercurial - https://www.ibm.com/developerworks/aix/library/au-mercurial/

[28] Mercurial vs git technical aspects - https://www.atlassian.com/blog/software-teams/mercurial-vs-git-why-mercurial

[29] Bitkeeper - http://www.bitkeeper.com/

[30] Bitkeeper wikipedia - https://en.wikipedia.org/wiki/BitKeeper

[31] Bitkeeper github - https://github.com/bitkeeper-scm/bitkeeper

[32] Bitkeeper opensource - - https://news.slashdot.org/story/16/05/10/1840255/11-years-after-git-bitkeeper-is-open-sourced

[33] Linus torvalds and bitkeeper - http://www.infoworld.com/article/2670360/operating-systems/linus-torvalds--bitkeeper-blunder.html

[34] Linux and bitkeeper - https://www.linux.com/news/bitkeeper-and-linux-end-road

[35] Stevens, M., Bursztein, E., Karpman, P., Albertini, A., & Markov, Y. (2017, August). The first collision for full SHA-1. In Annual International Cryptology Conference (pp. 570-596). Springer, Cham.

[36] Perforce vs git - - http://www.infoworld.com/article/2955650/development-tools/git-isnt-good-enough-version-control-enterprises.html

[37] Best version control for a standalone computer - http://stackoverflow.com/questions/138621/best-version-control-for-lone-developer

[38] G2crowd best version control - https://www.g2crowd.com/categories/version-control-systems

[39] Choosing the best version control - https://www.codeproject.com/Articles/431125/Choosing-a-Version-Control-System-A-Beginners-Tour

[40] Version control options - https://www.sitepoint.com/version-control-software-2014-what-options/

[41] Rhodecode survey - https://rhodecode.com/insights/version-control-systems-2016

[42] Popular version control systems - https://www.projecthut.com/version-control-systems-compared/